Proceedings of the 52nd Hawaii International Conference on System Sciences | 2019

Automated Testing of Motion-based Events in Mobile Application

Seyedeh Sepideh Emam
University of Alberta
emam@ualberta.ca

James Miller
University of Alberta
jimm@ualberta.ca

Abstract

Automated test case generation is one of the main challenges in testing mobile applications. This challenge becomes more complicated when the application being tested supports motion-based events. In this paper, we propose a novel, hidden Markov model (HMM)-based approach to automatically generate movement-based gestures in mobile applications. A HMM classifier is used to generate movements, which mimic a user's behaviour in interacting with the application's User Interface (UI). We evaluate the proposed technique on three different case studies; the evaluation indicates that the technique not only generates realistic test cases, but also achieves better code coverage when compared to randomly generated test cases

1. Introduction

Embedding hardware devices, such as movement sensors (accelerometers and gyroscopes), in mobile devices complicates testing procedures. Users are able to interact with the application by touching, tilting, shaking, and rotating the mobile devices. When a device is in motion or its screen is continuously touched, the probability of unintentional inputs increases; in such circumstances, automatically generated test suites are needed to produce accurate test cases and accelerate the mobile application testing procedure. Tools and techniques have been developed to test the quality of mobile applications, but the number of approaches that focus on automated testing is very limited. The majority of these automated testing tools offer capture-and-replay functionality to test the application's User Interface (UI).

Writing and continually improving motion-based test cases is a difficult task when testing mobile applications that use movement-sensor data. Therefore, considering existing mobile testing tools and approaches, two problems exist: 1) no automated approach is provided; and 2) generating test cases for motion-based mobile applications remains unconsidered. Thus, we propose a new approach to address these limitations. It is argued that mimicking user behaviour is one of the key factors in generating gesture-based test cases. It helps in executing realistic test scenarios and standard gestures [1], [2].

We propose a novel approach, which synthesizes the motions, and subsequently, simulates the test cases based upon the formalized gestures. Motion data is represented by the data captured, using the movement sensors and the objects' positions (2D coordinates) on the screen. An application can then use the sequences of motions to simulate the gestures and test the UI. To increase the chance of generating realistic movements, a set of training data is generated by human users and is used to train hidden Markov model (HMM) classifiers; these models are iteratively used to generate new motion sequences. Gestures and animations are commonly considered to be the key components in modern mobile user interface design; hence this work directly targets the heart of the matter in this new and evolving application domain.

In summary, the generated motions are used to automatically produce test cases, mimicking human-generated gestures with the technical goal of increasing code coverage. This study contributes to the research in this area by:

- Proposing a new approach to synthesize motion data, and make it executable as a test input to the application being tested.
- Applying a HMM classifier on the training data to create a set of HMMs, and subsequently using them to generate motion sequences.
- Evaluating the effectiveness of the proposed approach in terms of, (1) mimicking the user's behavior; and (2) increasing the code coverage of the software under test (SUT).

This paper is organized as follows. Section 2 provides background information on mobile applications, particularly motion-based gesture testing. Section 3 describes an overview of the proposed approach, the gesture synthesis and simulation

URI: https://hdl.handle.net/10125/60181
ISBN: 978-0-9981331-2-6

procedures, while Section 4 provides the design and implementation details. Section 5 provides a running example of the proposed test case generation approach. Section 6 discusses the evaluation phase, experimental setup, and results. Finally, Section 7 presents the conclusions.

2. Background and Literature Review

The growth in developing mobile testing procedures and techniques has been insufficient. Although many testing methods and tools exist for desktop and server/host software, most of them are not applicable for testing "mobile software" [3]. Although many traditional testing tasks are common between mobile applications and the desktop/web-based applications, several key factors cause challenges in the mobile testing procedure. Mobile devices are different in terms of screen sizes, platforms, input methods, and the quality of the sensor data. Such differences can easily multiply testing efforts. This can easily affect the quality of the application, along with the time of the marketplace and the costs of construction. Integrating automation approaches with test case generation procedures is a key factor in addressing these issues in the "mobile testing era", where many test cases need to be executed on a large selection of mobile devices and configurations.

In this regard, [4] presents a framework to test the functionality of mobile applications when a device is moved to a new network. The framework uses an application-level emulator to transfer the application across networks to ease the testing process under different network technologies. Additionally, [5] suggests a quality assurance framework to define key patterns and metrics in mobile application testing. Although these studies provide insights into the testing of mobile applications, they do not cover the test case generation phase. Several studies focus on automated testing for mobile applications have also been conducted; [6]–[9] suggest different, automated, graphical user interface (GUI) testing approaches for Android applications.

To test the GUI, the mobile application needs to be executed with user interaction events. With technological advancement in smartphones and tablets, natural user interfaces (NUIs), which no longer use keyboards and keypads as human-machine interfaces, have become popular. Touch-sensitive screens, speech recognizers, and gesture detectors are the primary interaction channels in the new generation of mobile applications. This era of application testing is relatively new, and only a limited number of studies have been performed to address these testing challenges [9], [10].

Mobile applications, which allow users to control the applications' functionality through NUIs, normally recognize gestures by using the data provided by the embedded sensors in the mobile device [11]. Several smartphones and tablets contain accelerometers to control motion inputs. One of the most common applications of accelerometers is presenting the landscape and portrait views of the screen based on the way the device is being held. The 3-axis model of the accelerometer is able to measure the magnitude and direction of the acceleration (gravitational force) as a vector $[ax_k, ay_k, az_k]$ for a motion k in a 3D space. Combining all three accelerations, lets the application detect the device's movement in any direction and obtain the device's current orientation. Depending on the graphical capabilities of mobile applications, 2D or 3D versions of the acceleration vector are considered. From the tester's perspective, testing applications that support motion-based events introduce a new complexity to the testing procedure; motion-based gestures should be accurately specified and reliably reproduced [9]. The lack of formal motion-gesture specification prevents testers from developing an automated test generation approach. The next section presents the simulation and synthesis procedures of such motion-based events.

3. Gesture Simulation

In the simplest process, test data-points can be provided by using a random test generation approach, which randomly creates data frames within a defined range to move the object on the screen. It can be expected that the number of reasonable gestures, which are created randomly are very limited. Therefore, even if these test cases are able to cover an acceptable number of branches in the source code, they may not be able to reveal faults a human user can discover simply because they cannot replicate standard gestures.

This study considers an automated test case generation procedure for applications interacting with users using motion-based events. Users normally interact with these applications by performing a sequence of gestures, e.g. by moving a flying or bouncing object on the screen or drawing geometrical shapes by touching the screen. User-generated gestures are transferred to the object or touched location to move the object toward the desired direction or to draw a geometrical shape (e.g. circle) around the touched point. It is noteworthy that motion-based events are not only used to move an object on the screen; sometimes, shaking a mobile phone in a specific direction or touching and dragging the screen leads to executing a function or opening another application. This study

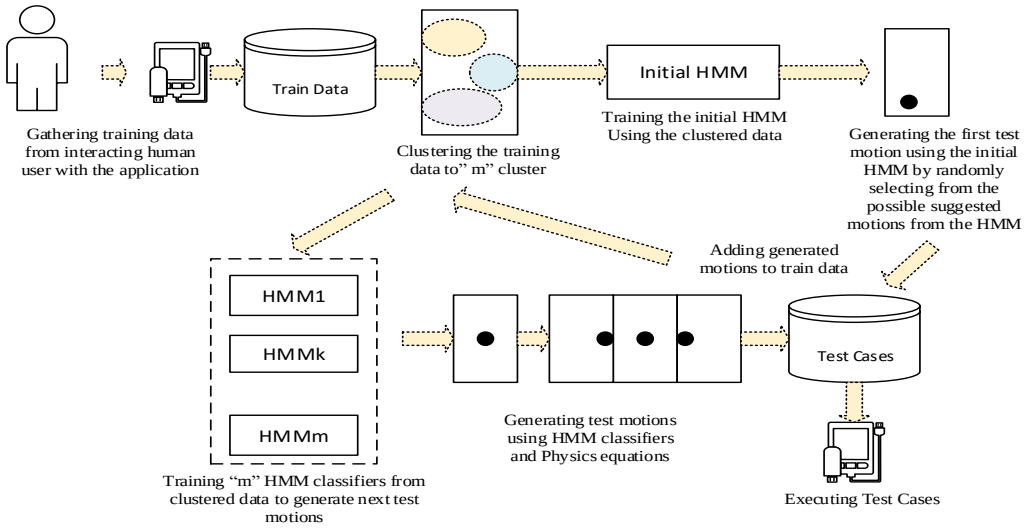

Figure. 1. An overview of applying the proposed approach on the application with flying object. It consists of both training the initial HMM (top) and test generation process using HMM classifiers (bottom)

focuses on the procedure to automatically generate test motions on both types of applications: (1) applications with flying object(s) (two case studies); and (2) applications with a touch-sensitive screen (one case study). In such cases, several parameters can affect a single motion (such as the object size, the size of the screen, an object's location, etc.). Since users are free to touch, move and shake their mobile phones in any desirable direction and speed, a testing approach must be able to generate sets of standard gestures, which are not only executable on the application but also resemble the human-generated motions.

The proposed technique contains several steps and details, which are depicted in the framework provided in Figure 1. The proposed approach consists of the following steps:

1. Gathering training data: A user interacts with the application and generates motions to be used as a training set. (It is worth noting that the person is not instructed to generate any specific motions and the generated motions are the result of a volunteer interacting with the application for the first time.)

2. Clustering motions: the k-means clustering algorithm is used to identify the relationship between data points (motions) generated. It is well known that data clustering is a successful approach in recognizing and categorizing human expressions, gestures and actions. More specifically, the motion parameters are partitioned into k clusters, such that each motion is allocated to the cluster with the nearest mean.

3. Training Initial HMM: In order to produce the first gesture, an initial HMM is trained using human-generated motions. As we utilize time-varying motion sequences, HMMs can be used to model human skills such as interactions with mobile applications. Using the expectation-maximization (EM) algorithm the initial HMM trains a model, where its hidden states indicate motions' clusters, generated in the first step. The probability of a gesture belonging to a specific cluster (state) is estimated and used to calculate the first motion acceleration parameters. The first motion's acceleration is calculated by computing the mean of the accelerations in each HMM state and by selecting one pair randomly. Hence, we can hypothesize that the test sequence produced can potentially mimic human generated gestures.

4. Generating the test data using HMM classifiers: We apply HMM classifiers on clustered data to generate test motions. For each cluster, the dynamics of each motion class is learned with one HMM. Thus, having m motion-clusters, m HMM classifiers need to be applied. HMM classifiers classify each motion as a function of a future time frame [12]. Thus, the probability of a test case belonging to each cluster is calculated using the Forward algorithm [11]. The motion-cluster with highest Forward probability is selected and the mean of the acceleration of the motions belong to this cluster is considered as the next motion's acceleration.

5. Adding generated motions to the training set: in order to avoid over-fitting the model, the motions should be added to the training set. This helps the model to learn from the data rather than memorizing the trend.

6. Storing and executing test cases: Once, for example, the ball hits the vertical wall the set of test motions generated, since the last hit, are stored as test cases and will be used to generate real motions.

3.1 Synthesizing Motion Sequences

This section describes the method of instantiating motion sequences for complicated motion-based applications, which transfer the users' gestures to a bouncing object. However, the application of this approach is not limited to events using sensor-generated data; it can be easily used to generate automated test cases for any type of motion-based events. Following the previous section, two sets of data (motion sequences) are considered in this study:

- The training data, which is captured during a real user's interaction with the application and is used to train the initial HMMs.
- The second set is the test data, which is generated by using the test generation algorithm and is presented to the application being tested to evaluate its functionality. To create meaningful test data, which is recognizable by the trained HMM and its corresponding classifier, we describe a single motion k by a 6-tuple $(lx_k, ly_k, vx_k, vy_k, ax_k, ay_k)$, where lx_k, ly_k indicates the object's location, vx_k, vy_k determine the velocity, and ax_k, ay_k describe the acceleration of the motion in 2D space at a specific time interval. Figure 2a shows the 3D acceleration axes on a smartphone, which also contains a z-axis. In order to simplify the explanation of the algorithm and cover more common applications.

This study also considers two time intervals during the test generation procedure:

- The first time interval happens every φ ms [7] to capture information regarding the current motion and position of the object and to calculate the next motion using SUVAT equations [13], [14].
- The second time interval happens every θ ms, which is estimated by selecting the minimum possible time between two gestures, generated by human users. Hence, the estimation of θ assists the algorithm to generate more realistic (complex) gestures as it accounts for the limitations of kinematics.

Figure 2b shows a gesture consisting of a sequence of motions happening within these two intervals. Each sequence of motions is terminated by the occurrence of a specific condition in the application being tested; for example, when the flying object hits another object.

Definition1: A test case (TC) consists of a set of motions $(M = \{m_1, \dots, m_n\})$,where $m_{k \le n}$ is a 6-tuple $(lx_k, ly_k, vx_k, vy_k, ax_k, ay_k)$. The number of tuples (motions) in each TC depends on the number of detectable motions before the termination condition.

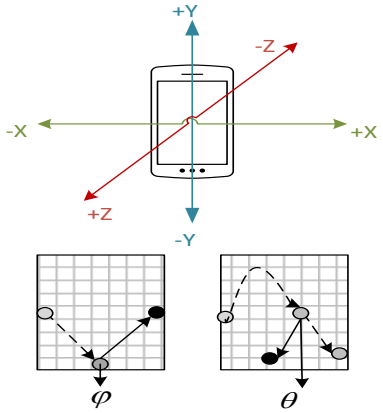

Figure. 2. (a) screen after hitting the edge in first time-interval φ; (right) the 3D acceleration axes on smartphones; and (b) a gesture containing a sequence of motions happening within two intervals: (left) a bouncing object moving in the proposed approach calculates the next movement after the second time-interval θ

4. HMM-based Test Case Generation

The clustering algorithm is applied to groups of motions with similar behaviour and allocates them into a single cluster. These clusters will be used as the class labels for the HMM classifiers. This means that each class indicates a set of similar motions in the corresponding cluster. The clustered data will be used to train an initial Hidden Markov Model. The HMM in this study is characterized by the following elements:

- a set of latent states $S = \{s_1, s_2, \dots, s_L\}$, which are hidden from the external observer and indicates the class of motion sequences;
- a set of observable states $V = \{v_1, v_2, \dots, v_N\}$, where each is mapped to a corresponding motion sequence (m_k);
- a transition probability $[A]_{ij} = \{a_{ij}\}$,

$a_{ij} = P(Q_{t+1} = s_j | Q_t = s_j), 1 \le \text{i,j} \le \text{L}$, which determines the transition probability between different classes. For the initial modelling process, because human users generate the motions, the initial transition probabilities between different classes of motions can be extracted directly from the training data;

- an emission probability $[B]_{jk} = \{b_{j\,(vk)}\}$,

$b_j\,(v_k) = P(M_t = v_k | Q_t = s_j), 1 \le \text{j} \le \text{L}, 1 \le \text{k} \le$ Nwhich indicates the probability of a motion sequence belonging to a specific class (estimated by frequency counting on the clustered training corpus); and

- initial state distribution, $\Pi = \{\pi_i\}$,

$\pi_i = P(Q_1 = s_i), 1 \le \text{i} \le \text{L}$. Each and every state can be an initial state in this study.

Using the values of A, B, and Π, an HMM can be used as a generator to create an observation sequence

(where T is the number of motions in the test case): $M = \{M_1, M_2, M_3, \dots, M_T\}$. This initial HMM model is used as an input to an expectation-maximization (EM) algorithm. This algorithm estimates the optimal model with the highest likelihood of the estimated parameters. In algorithm 1, this procedure is done by running the HMM function in the first line. Then, the initialAccel function initializes, the acceleration parameters of the first test motion by calculating the mean of the acceleration pairs in each HMM state and by selecting one pair randomly. Then, in lines two and three of this algorithm, the CreateMotion function generats a motion sequence using the SUVAT equations and the Update function stores the newly created motion sequence as the current motion. After generating the initial motion, the CreateMotion and Update functions are called again but this time within the time interval φ, until a termination condition happens (line 4-8). This procedure generates a simple gesture based upon the previous motion, using appropriate physics equations. In order to generate more realistic and complicated gestures, we propose using the HMM classifier to detect the sequence class label at each interval θ.

The HMMClassifier function in line 10 of the algorithm classifies the current motion sequence into an appropriate class of gestures. This function combines a set of sequences of motions and a list of class labels to train one HMM per class label (where L is the number of class labels). Subsequently, the trained models are used to calculate the forward probability of a motion sequence per model. The forward algorithm computes the forward probability,$\alpha_k(t)$, as the joint probability of observing the first t vectors $m_t, T = 1, \dots, t$ while in state k at time t. Given a list of forward probabilities, we are able to select a model with the maximum probability and assign its corresponding class label as the motion's class label and estimate the next motion values by calculating the mean of the accelerations of the motions (the Accel function in line 10). Moreover, the generated motion is added to the training set to avoid over-fitting. This helps the model to learn from the data rather than memorizing the trend (line 14).

Putting it all together, lines four to fourteen of Algorithm 1 create a set of motion sequences within two different intervals. Simple gestures are generated based on physics equations once the first time-interval happens; more complicated motions (e.g. gestures with variable accelerations) that may require a longer time period to be created by a human user are generated within the second time interval. An example of a simple motion is the one calculated by the SUVAT equations after the bouncing ball hitting the horizontal wall. While the complex one is a motion calculated by HMM classifiers for a ball slowly bouncing in the middle of the screen.

ALGORITHM 1. TEST CASE GENERATION PROCEDURE FOR CASES WITH ACCELERATION INVOLVED

Input: *Initial position of the bouncing object (x,y), training data set (S), set of class labels (C);* $i = 2$*;*
Output: *Test case (TC)*

1. (ax,ay)⟵ initialAccel(*HMM(S,C)*)
2. m_1 ⟵CreateMotion(ax,ay,x,y)
3. Update(ax,ay,x,y)
4. **While** *(!terminalCondition)*
5. ***if*** $(curTime - lastUpdate1) \geq \varphi$)
6. $i \leftarrow i + 1$
7. m_i ⟵CreateMotion(ax,ay,x,y)
8. Update(ax,ay,x,y)
9. $S \leftarrow S \cup \{m_i\}$
10. ***if*** $(curTime - lastUpdate2) \geq \theta$)
11. (ax,ay)⟵Accel(*HMMClassifier(m_i,S,C)*)
12. $i \leftarrow i + 1$
13. $m_i \longleftarrow$ CreateMotion(ax,ay,x,y)
14. Update(ax,ay,x,y)
15. $S \leftarrow S \cup \{m_i\}$
16. **End while**
17. **Return** $TC \leftarrow \{m_1, \dots, m_i\}$

**lastUpdate1 indicates the last update that happened at interval φ while lastUpdate2 indicates the last update that happened at interval θ*

5. Running Example

In order to clarify the proposed test case generation procedure, we consider a very small portion of the training data generated by a human user in the bouncing ball application. An example of a single motion is provided below:

05-07 17:36:15.828: Vx(32065): -2.7148619
05-07 17:36:15.828: Vy(32065): -2.7148619
05-07 17:36:15.828: lBallX(32065): 549.0
05-07 17:36:15.828: lBallY(32065): 20.0
05-07 17:36:15.828: Ax(32065): 0.090979666
05-07 17:36:15.828: Ay(32065): -0.1233013

In this running example, we follow the test generation framework (Figure 1) step by step to generate test cases:

1. Gathering training data: 30 motions in the format of 6-tuple $(lx_k, ly_k, vx_k, vy_k, ax_k, ay_k)$ are gathered as the result of user interaction with the application.

2. Clustering motions: the training data is clustered into 2 distinct clusters (classes) using the k-means algorithm. Due the space limitations. a partial view of the clusters are provided in Table 1.

3. Initial HMM training: the clustered data is then used to train the initial HMM using Baum Welch algorithm. In this case, the HMM model contains 30

observable states and 2 hidden states (since there are only two clusters). Then, the acceleration parameters of the first data motion are generated by calculating the mean of the acceleration pairs of the motions belonging to each hidden state of the initial HMM and subsequently selecting one pair randomly. After determining the initial acceleration parameter, the first motion is created:

(1) initial acceleration parameter:

$$(ax, ay) = (0.59855044, -0.91578215)$$

(2) the initial location of the ball in the screen:

$$(lx_0, ly_0) = (309,253)$$

(3) knowing that the initial velocity is equal to zero (ball is not moving at the beginning):

$$(vx_0, vy_0) = (0,0)$$

motion $m_1(309.080798,252.876369,0.1755132,-0.2747346,0.059855044,-0.091578215)$ is generated using physics equations: $v = at + v_0$ and

$$l = l_0 + v_0 t + \frac{1}{2} a t^2$$

Then within the time interval $\varphi = 300ms$ other motions are also generated through the same process with the difference that the acceleration of the current motion is used as the initial acceleration for the next motions. These motions will be added to the training set to avoid over-fitting. (Figure 3 depicts a schema of the trained initial HMM).

4. Test data generation using HMM classifiers: Now, in order to generate more complex motions (within time interval $\theta = 500\,ms$), two (number of classes) HMM classifiers are trained and the forward probability of the current motion is calculated to reveal the class of motions it belongs to. Then, the mean of the accelerations of the motions belonging to this class are calculated; and again, are used as input of the motion equations to calculate the velocity and location parameters. For example, if the occurrence likelihood (forward probability) of the current motion $m_i(20, 492.07,2.1625056,2.1625056,-0.00778115,0.24600422)$ in class c_2 reaches the maximum amount compared to the other class (c_1), the mean of the acceleration of the motions in class c_2 is calculated and will be used as the new current motion's acceleration. In this case, the mean of the accelerations in $c_2$in equal to (0.3471,1.1162). Therefore, using physics equations, the next motion would be:

$$m_{i+1}(21.1246403,493.2907778,2.3360556,2.7206056,0.3471,1.1162),$$

This motion also will be added to the training set. Once, the ball hits the vertical wall, the motions generated since the last hit, are saved in the form of a test case and will be executed to move the ball.

6. Empirical Evaluation

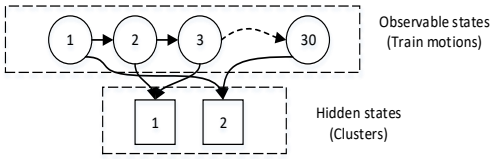

Figure. 3. An overview of trained HMM in running example

To study the proposed approach, we performed an experiment on three case studies; we attempt to answer the following research questions:

- Can the test-generated motions mimic actual user behaviour?
- Does the proposed method improve the code coverage of the SUT when compared to existing automated techniques (random testing)?

The first case study is an Android application, a bouncing ball application, designed to record a data set of coordinates from shake and tilt gestures performed by human users *(LOC=716).* This application contains one flying object (round ball), which bounces on the screen; the ball moves by processing information it captures from a mobile devices accelerometer. The dynamics of a bouncing ball follows a set of physics laws and equations [30], which are used in this study. Since covering the details of such equations is beyond the scope of this research, we only discuss some of the case-specific motions and equations:

- When the application starts, the ball is stable in a corner of the screen, waiting for a motivation. Depending on the power of the first motion, the ball starts moving toward the motion's direction. In this study, the time interval φ is fixed at 300 milliseconds, following [9], [10] to capture the information regarding the current motion and position of the ball on the screen and to calculate its next position.
- The second time interval θ is equal to 500 milliseconds because the time windows between gestures created by users it varies from 500 milliseconds to one second, we select the lower bound to create standard motions.
- Each sequence is terminated whenever the ball hits the vertical edges of the screen.

Table 2 (First two columns) indicates the simplest possible actions that can be performed in this application, along with their corresponding gestures. It is noteworthy that in designing this table, it is assumed that the ball has enough space to move toward each direction. Obviously, it cannot for example move to the left when it has already hit the right-side edge. Any combinations of these actions (e.g. curving), which may be produced by rotating, tilting the device. For

example, when the user rotates or tilts the mobile phone toward the right, the ball can moves in a curve instead of moving in a straight line to the right.

The second case study is another android application called Bubbles, which is able to draw circles around the touched points on the screen *(LOC=423).* In order to generate circles (bubbles), the user touches or pushes the screen resulting in a circle being gradually grown from the touched point. The maximum length of the circle's radius is predefined and fixed, so the circle keeps growing until its radius is equal to the maximum number or the user touches another point in the screen. Table 2 (Second two columns) shows the action (motion event) and its corresponding gesture. The sequences of motions are continuously generated until a border is touched. Then, the generated set is considered as a test case.

In order to evaluate the performance of proposed test case generation approach in a more complex framework, we modified the Bouncing ball application by adding a second more flying object. The second ball behaves the same as the first one (Table 2 – First two columns), except for the difference that its initial location in the bottom right-hand corner (the original ball is located in the left side), thus depending to the amount of acceleration received from the sensors, they can move in diverse directions. The same test generation algorithm is used to produce test cases for the extended Bouncing ball application (*LOC= 1054*) and test motions are stored in two separate sets of test suites for each ball.

6.1 Experimental Results

To answer the research questions and evaluate the efficiency of the proposed test generation approach, volunteers interacted with the applications and produced motion sequences which are then used as training sets. In the Bouncing ball application, a set of training data was obtained by recording the motion coordinates for three minutes from a total of 317 gestures performed on two different Android devices. Applying the silhouette score, we grouped the motions into 95 clusters. For the extended version of this application, 600 motions and 105 clusters were considered. This data is recorded in 6 minutes. For the Bubble application, these numbers were 481 and 95 respectively (motions are stored for 2 minutes). The amount of time allocated to each training process is estimated based upon the time a new user needs to become visually familiar with the application and to generate a set of motions. In this study, this time is estimated by calculating the mean of the time that new users require to generate a reasonable set of motions for the considered applications.

To evaluate the quality of the generated test cases in all case studies, 20 sets of 200 motion sequences were generated using the proposed technique. In addition, for the Bouncing ball application, the same number of motion sequences (20 sets of 200 motions) was created by random test generator procedures:

- Algorithm 3: takes a human-user motion to initialize the acceleration or position parameters then creates the further motions based on the current one by randomly selecting a physics equation.
- Simple Random Algorithm: Creates test cases by simply generating random motion sequences within the data ranges supported by the hardware. In this study, a human user also generates the initial motion. Since, the HMM-based technique is using human-data to train the initial model and generate the first motion, the simple random test case generation process also get initialized by human-generated data.
- Hybrid approach: In order to conduct a fair comparison, some experiments have been designed to execute combinations of human and randomly generated test cases (e.g. "Human + Simple random" and "Human + Algorithm 3"). This means that using human data is not limited to the initialization phase and user-generated data forms half of the test cases. Therefore, a hybrid test case consists of a combination of human generated motions and random motions.

Since the acceleration parameter and its corresponding physics equations are not considered in the second case study, only the simple random algorithm is implemented to generate the random touched-points.

To answer the first research question, we classified test cases by using the HMM classifier. Then the occurrence likelihood (LC) of each sequence of motions for each class label are calculated where $\{LC = P(M|\Lambda_i), \Lambda_{i\leq L} and\ M \in TC\}$, where L is the number of classes. In this case, when $\max_{L} P(M|\Lambda_i)$ is a small quantity, it can be concluded that the test case TC is not behaving similar to the test cases that were used to create the classes. Additionally, since these classes are created using human-generated motions, it can be implied that the probability of the test case TC being generated by a human user is low.

The results show that the motions generated using the HMM-related technique have a higher forward probability (occurrence likelihood) compared to other approaches. Accordingly, it can be concluded that the test cases generated using the proposed technique are more likely to be generated by a human user. The reason is that each class label describes a set of human-generated motions; therefore once a motion has high occurrence likelihood in one of these classes, it can be

concluded that the probability of being generated by a human user for this motion is high.

To address the second research question, the JaCoCo code coverage library was used. Using this toolkit, bytecode instrumentation is applied, and the branch coverage value is measured. Since we generated 20 sets of 200 test cases using each approach, the means of the coverage percentages on all sets, are calculated to achieve more accurate results (In total, 64000 motion sequences are generated during the experiments). Table 3 reports the means of the branch-coverage percentages calculated by running each of the test case generation approaches. The result of applying the Wilcoxon signed rank test indicates the HMM-based approach is significantly different from the other techniques in terms of code coverage. Table 3 also reports the p-values and delta estimates at the 95% confidence interval. The Cliff's Delta measure provides more detailed information to this picture by showing that a "large" effect size exists (in favour of HMM-based approach) for all of the comparisons. The achieved results confirm that the HMM-based test case generation approach not only automates the test generation and execution procedure for motion-based events, but also (1) creates better test cases in terms of mimicking actual user gestures; and (2) improves the (branch) code coverage for the SUT (Table 4).

Table 1. Simplest Supported Actions and Gestures in Both Types of Application

Cluster 1	***Cluster 2***
(211.36267, 502.0, 9.787367, 9.787367, -0.30645782, 7.948151)	(20.0, 344.4505, 24.511274, 24.511274, -4.5637164, -3.260261)
(220.37634,502.0, 9.259302, 9.259302, 0.55006784, 7.753622)	(20.0,45.516983, 8.89769, 8.89769, -6.5458517, 6.4084578)
(229.64267,502.0, 8.681731, 8.681731, -0.55006784, 7.753622)	(26.236174,20.0,3.8991215,3.8991215,11.504282,5.4646373)
(245.16068,502.0,7.443665,7.443665, -0.8355764, 7.9068513)	(20.0,182.77194, 23.407976, 23.407976,8.195976, -7.834369)
(252.28542,502.0, 6.5663095, 6.5663095,-0.8355764, 7.9068513)	(20.0,235.21193, 19.96578, 19.96578, -8.742604,0.1829845)
(270.06726,502.0, 2.6941133, 2.6941133, -1.039682, 7.953538)	(300.0,118.05718, -28.868063, -28.868063, -8.03005, -4.4044623)
(272.01288,502.0, 1.5729084, 1.5729084, -1.067814, 7.89907)	(300.0,367.61127, -36.233944, -36.233944, 8.330351, 1.1209484)
(271.1313, 502.0, -1.6668775, -1.6668775, -1.1701661, 7.817667)	(20.0,378.44443, 28.222904, 28.222904, -2.80235, -0.7510143)
…	…

Table 2. Simplest Supported Actions and Gestures in Both Types of Application

Bouncing Ball / Extended Bouncing Ball		***Bubbles***	
Action	***Gesture***	***Action***	***Gesture***
Tilt the device toward left.	The ball bounces to the left side of the screen.	Touch/Push the screen.	The circle is drawn around the touched-point.
Tilt the device toward right.	The ball bounces to the right side of the screen.		
Tilt the device to the front.	The ball bounces down.		
Tilt the device to the back.	The ball bounces up.		

Table 3. Results of Calculating Effect Size Measure and the Mean of Code Coverage For Test Case Generation Methods in All Case Studies

	Approach	***Mean of Code Coverage (%)***	***Approach***	***Delta Estimate***	***p-value***
Bouncing ball	HMM-based	79.26	HMM-based Vs. Algorithm 3	-0.965	4E-05
	Algorithm 3	55.95	HMM-based Vs. Simple Random	-1	4E-05
	Simple Random	33.05	HMM-based Vs. Human + Algorithm 3	-0.7357	0.00019
	Human + Algorithm 3	63.63	HMM-based Vs. Human + Simple Random	-0.6761	0.00034
	Human + Simple Random	62.73	HMM-based Vs. Human	-0.7225	0.00017
	Human	60.2			

Extended Bouncing ball	HMM-based	81.3	HMM-based Vs. Algorithm 3	-0.95	1.9E-06
	Algorithm 3	52.75	HMM-based Vs. Simple Random	-0.95	1.9E-06
	Simple Random	31.77	HMM-based Vs. Human + Algorithm 3	-0.71	3.4E-05
	Human + Algorithm 3	62.78	HMM-based Vs. Human + Simple Random	-0.575	0.0028
	Human + Simple Random	62.98	HMM-based Vs. Human	-0.62	0.0002
	Human	62.05			
Bubbles	HMM-based	92.06	HMM-based Vs. Human + Simple Random	-0.9325	5E-05
	Simple Random	74.28	HMM-based Vs. Human	-0.9325	4E-05
	Human + Simple Random	79.97	HMM-based Vs. Simple Random	-1	4E-05
	Human	78.94			

Table 4. Results of Providing Same Resources as HMM-based to Random

	Approach	***Code Coverage (%)***	t_g***(min)***	t_e***(min)***
Bouncing ball	HMM-based (200 motions)	75%	0.5	3
	Random (33800 motions)	42%	0.17	23
Extended Bouncing ball	HMM-based (200 motions)	75%	0.5	3.2
	Random (33800 motions)	40%	0.17	24
Bubbles	HMM-based (200 motions)	92%	0.2	1.2
	Random (33800 motions)	78%	0.08	15.6

7. Conclusions

Testing mobile applications that use motion-based gestures to interact with users poses a new challenge. Test inputs should be realistic motion sequences, which are able to simulate the user's behaviour in interacting with the application. This helps in revealing defects, which remain unknown in applications because they do not conform to expected human-generated motions. Since, Markovian models have been successfully used in software testing studies to generate models representing common user behaviour in UI testing.

In this paper, we have proposed a new HMM-based approach, which presents a solution for automating the testing process for applications supporting motion-based events. Using this method, gestures can be formally specified as sequences of motions, which are easy to re-execute in the application. Therefore, an HMM classification approach is used to classify the current movement into a class of motions providing the best description of the gesture's characteristics. Then, according to the results provided by the classification approach and using standard movement equations, a realistic proxy for the likely next movement coordinates can be estimated.

We evaluated our approach by generating a set of test inputs for three Android applications with a gaming theme. The empirical results show that the generated test cases using HMM-based approach not only cover a higher number of branches in the source code compared to randomly generated test cases, but the occurrence likelihood of the corresponding motion sequences in model trained by user generated data is also higher in HMM-based approach. This indicates that the new approach outperformed the random method in generating test cases that mimic human-user behaviour.

8. References

[1] M. Hesenius, T. Griebe, S. Gries, and V. Gruhn, "Automating UI Tests for Mobile Applications with Formal Gesture Descriptions," in *Proceedings of MobileHCI'14*, 2014, pp. 213–222.

[2] C. J. Hunt, G. Brown, and G. Fraser, "Automatic testing of natural user interfaces," *IEEE 7th Int. Conf. Softw. Testing, Verif. Valid.*, pp. 123–132, 2014.

[3] B. Kirubakaran and V. Karthikeyani, "Mobile application testing — Challenges and solution approach through automation," in *2013 Int. Conference on Pattern Recognition, Informatics and Mobile Engineering*, 2013, pp. 79–84.

[3] I. Satoh, "A testing framework for mobile computing software," *IEEE Trans. Softw. Eng.*, vol. 29, no. 12, pp. 1112–1121, 2003.

[5] D. Franke and C. Weise, "Providing a software quality framework for testing of mobile applications," in *4th IEEE Int. Conf. on Softw. Testing, Verif., and Valid.*, 2011,

pp. 431–434.

[6] D. Amalfitano, A. R. Fasolino, P. Tramontana, S. De, U. Federico, and I. I. Napoli, “Using GUI Ripping for Automated Testing of Android Applications,” in *Proc. of the 27th IEEE Int. conf. on Automated Software Eng.*, 2012, pp. 258–261.

[7] C. Hu and I. Neamtiu, “Automating gui testing for android applications,” in *Proc. of the 6th Int. work. on Auto. of software test - AST ’11*, 2011, p. 77.

[8] C. D. Nguyen, A. Marchetto, and P. Tonella, “Combining model-based and combinatorial testing for effective test case generation,” in *Proc. of the 2012 Int. Sym. on Soft. Test. and Anal.*, 2012, p. 100.

[9] C. M. Prathibhan, A. Maliani, N. Venkatesh, and K. Sundarakantham, “An automated testing framework for testing android mobile applications in the cloud,” in *IEEE Int. Conf. on Adv. Comm. Control and Comput. Tech.* , 2014, pp. 1216–1219.

[10] R. N. Zaeem, M. R. Prasad, and S. Khurshid, “Automated generation of oracles for testing user-interaction features of mobile apps,” *IEEE 7th Int. Conf. Softw. Testing, Verif. Valid.*, pp. 183–192, 2014.

[11] J. An and K.-S. Hong, “Finger gesture-based mobile user interface using a rear-facing camera,” *IEEE Int. Conf. on Consumer Electronics*, 2011, pp. 303–304.

[12] O. Perez, M. Piccardi, G. Jesus, and J. M. Molina, “Comparison of Classifiers for Human Activity,” Lect. Notes in Comput. Science, 2007, pp. 192–201

[20] E. S. Choi, W. C. Bang, S. J. Cho, J. Yang, D. Y. Kim, and S. R. Kim, “Beatbox music phone: Gesture-based interactive mobile phone using a tri-axis accelerometer,” in *Proc. of the IEEE Int. Conf. on Industrial Technology*, 2005, pp. 97–102.

[12] C. B. Park and S. W. Lee, “Real-time 3D pointing gesture recognition for mobile robots with cascade HMM and particle filter,” *Image Vis. Comput.*, vol. 29, no. 1, pp. 51–63, 2011.

[13] S. O. Hara, Y. M. Lui, and B. A. Draper, “Unsupervised Learning of Human Expressions , Gestures , and Actions,” in IEEE Int. Conf. on Auto. Face & Gesture Recognition, 2011, pp. 1–8. [23] S. Gibet, N. Country, and J.-F. Kamp, *Lecture Notes in Artificial Intelligence*. 2005.

[14] D. Kleppner and R. Kolenjow, An Introduction to Mechanics. Cambridge University Press, 2013.

2007.